Neural Networks
Lecture 4: Radial Bases Function Networks

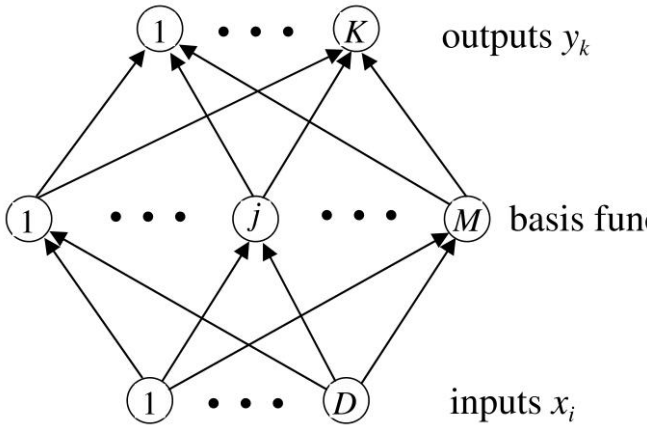
Farzaneh Abdollahi

Department of Electrical Engineering

Amirkabir University of Technology

Winter 2011

- Radial Bases Functions Networks (RBFN) is firstly proposed by Broomhead and Lowe in 1988
- Main features
  - They have two-layer feed-forward networks.
  - The hidden nodes implement a set of radial basis functions (e.g. Gaussian functions).
  - The output nodes implement linear summation functions (similar to MLP).
  - The network training is divided into two stages:
    1. The weights from the input to hidden layer are determined
    2. Then the weights from the hidden to output layer are found.
  - The training/learning is fairly fast.
  - RBF nets have better performance than MLP in some classification problems and function interpolation



- RBFN approximates $f(x)$ by following equation

$$f(x) = \sum_{i=1}^{n} w_i \phi(r)$$

where $r = \|x - c_i\|$

  - $x \in R^n$: input vector
  - c_i vector value parameter centroid (first layer weight)
  - w_i connection weights in the second layer (from hidden layer to output)
  - ϕ: activation function should be radially symmetric (i.e. if $\|x_1\| = \|x_2\|$ then $\phi(\|x_1\|) = \phi(\|x_2\|)$)

- Considering ϕ as Gaussian fcn: $\phi(r) = exp(-\frac{\|x-c_i\|^2}{2\sigma_i^2})$
  - σ_i: pos. valued shaping parameter (width)
  - Training RBFN is a process to find appropriate values of w_{kj}, c_{ij} and σ_j

Commonly Used Radial Basis Functions

1. **Linear Function**: $\phi(r) = r$
2. **Cubic Function**: $\phi(r) = r^3$
3. **Gaussian Function** $\phi(r) = exp(-\frac{r^2}{2\sigma^2})$
4. **Multi-Quadratic** $\phi(r) = (r^2 + \sigma^2)^{1/2}$
5. **Generalized Multi-Quadratic** $\phi(r) = (r^2 + \sigma^2)^{\beta}, \quad 1 > \beta > 0$
6. **Inverse Multi-Quadratic** $\phi(r) = (r^2 + \sigma^2)^{-1/2}$
7. **Generalized Inverse Multi-Quadratic** $\phi(r) = (r^2 + \sigma^2)^{-\alpha}, \ `\alpha > 0$
8. **Thin Plate Spline** $\phi(r) = r^2 ln(r)$
9. **Shifted Logarithm** $log(r^2 + \sigma^2)$

   where $r = \|x - c\|_2$

- The Gaussian and Inverse Multi-Quadric Functions are localized in the sense that $\phi(r) \rightarrow 0$ as $\|r\| \rightarrow \infty$
- For all the other mentioned functions: $\phi(r) \rightarrow \infty$ as $\|r\| \rightarrow \infty$
- In RBFNN the hidden layer and output layer play very different role.
- $\therefore$ It is appropriate to use different learning alg. for each:
  - First the hidden node centers are determined
  - Then the output layer weights are trained

Training RBFN: Basis Function Optimization

- One major advantage of RBFN is possibility of choosing suitable hidden unit without having perform a full nonlinear optimization of the whole network
- The methods of doing it are categorized to:
  - Unsupervised methods:
    are particularly useful in situations where labelled data is in short supply, but there is plenty of unlabelled data (i.e. inputs without output targets)
  - Supervised methods:
    get usually better results

Unsupervised Methods

1. Fixed center selected at random:
   - It is the simplest and quickest approach
   - Centers are selected as fixed M points randomly from N data points
   - Their widths are equal and fixed at an appropriate size for the distribution of data points.
   - The normalized RBF centered at c_j are defined

$$\phi_j(x) = exp(-\frac{M}{d_m^2}\|x - c_j\|^2) \quad c_i \subset x^p$$

   where d_m is max distance between chosen centers
   - The widths are $\sigma_j = \frac{d_m}{\sqrt{2M}}$
   - It ensures individual RBF's are neither too peaked nor too flat
   - For large training sets, this method provides reasonable results

Unsupervised Methods

2. K-Mean Clustering

- Using clustering techniques provides an improved approach which more accurately reflects the distribution of the data points.
- Partitions the data points x^p into K disjoint subsets S_j s.t. the sum of square error is min
- Each subset contains N_j data points based on min distance rule.
- K-Mean Alg:
  - 2.1 Choose a set of centers $c_1, ..., c_k$ arbitrarily
  - 2.2 Assign N samples to the K subsets using the min Euclidean distance rule:
    $x^p \in c_i$ if $\|x^p - c_i\| < \|x^p - c_j\| \quad \forall i \neq j$
  - 2.3 After all data points are assigned go to step 2.4
  - 2.4 Compute new subset center (c_i) s.t. min the cost function
    $J_i = \sum_{x \in S_i} \|x - c_i\|^2$
  - 2.5 If any subset center is changed, return to step 2.2, otherwise stop.

Supervised Training Method

- This method is generally giving better results than unsupervised procedures
- but the computational costs are usually enormous.
- Similar to MLP, this approach performs gradient descent on a sum squared output error function
- The error function would be

$$E = \sum_p \sum_k (y_k(x^p) - t_k^p)^2 = \sum_p \sum_k (\sum_{j=0}^{M} w_{kj}\phi_j(x^p, c_j, \sigma_j) - t_k^p)^2$$

where t_k^p: kth element of target vector in pth sample data, x^p: input vector of pth sample data, y_k is network output

- The weights/basis function parameters are updated as

$$\triangle w_{jk} = -\eta_w \frac{\partial E}{\partial w_{jk}}, \triangle c_{ij} = -\eta_c \frac{\partial E}{\partial c_{ij}}, \triangle \sigma_j = -\eta_\sigma \frac{\partial E}{\partial \sigma_j}$$

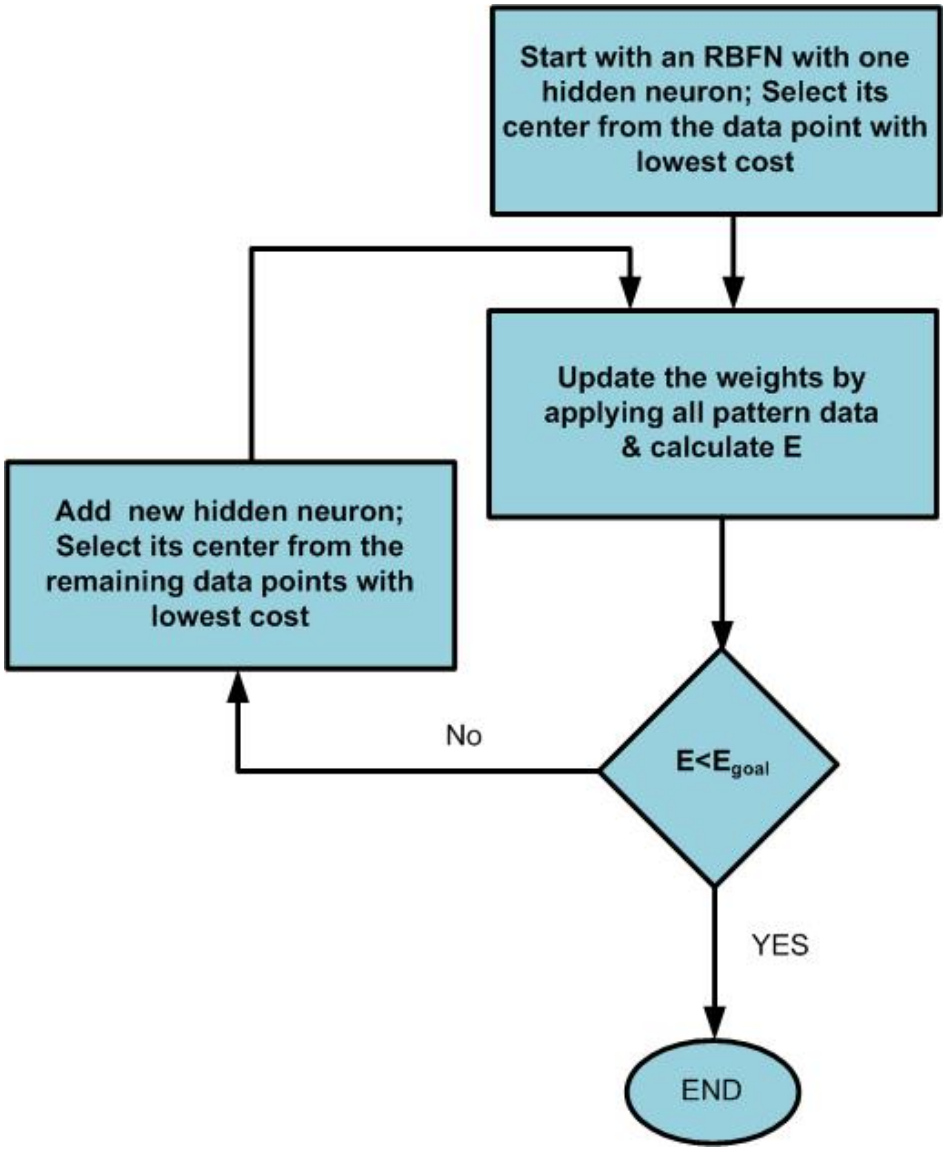
- The learning rates η should be selected carefully to avoid local minima and acceptable convergence rate and error
- **Finding the Output Weights**
- The output weights can be obtained by either supervised or unsupervised methods
  1. Unsupervised method:
     - After the input to hidden weights are found (centers), they are kept fixed for the second stage of training during which the hidden to output weights are learned
     - Since the second stage involves just a single layer of weights w_{jk}, they can easily be found analytically by solving a set of linear equations.
     - Usually, this can be done quickly, without the need for a set of iterative weight updates as in gradient descent learning.

Training: Finding the Output Weights

1. Unsupervised method cont'd:
   - Given the hidden units activation $\phi(x, C_{ij}, \sigma_j)$, the below series of simple linear equations should be found
   $$y_k(x^p) = \sum_{j=0}^{M} w_{kj}\phi_j(x^p) = t_k^p$$
   - This can be rewritten as $\Phi W^T = T$, where $W_{kj} = \{w_{kj}\},\ \Phi_{pj} = \{\phi_j(x^p)\},\ T_{pk} = \{t_k^p\}$.
   - Therefore
   $$W^T = \Phi^* T$$
   where $\Phi^* = (\Phi^T\Phi)^{-1}\Phi^T$ is pseudo inverse of Φ^T
   - In practice we tend to use singular value decomposition (SVD) to avoid possible ill-conditioning of Φ.
2. Supervised method: the weights of second layer can be trained by supervised learning method like BP alg2

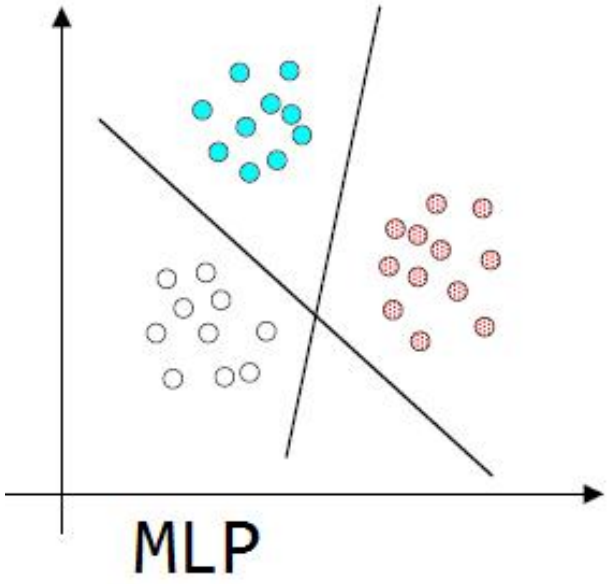
Training By Orthogonal Least Squares [1]

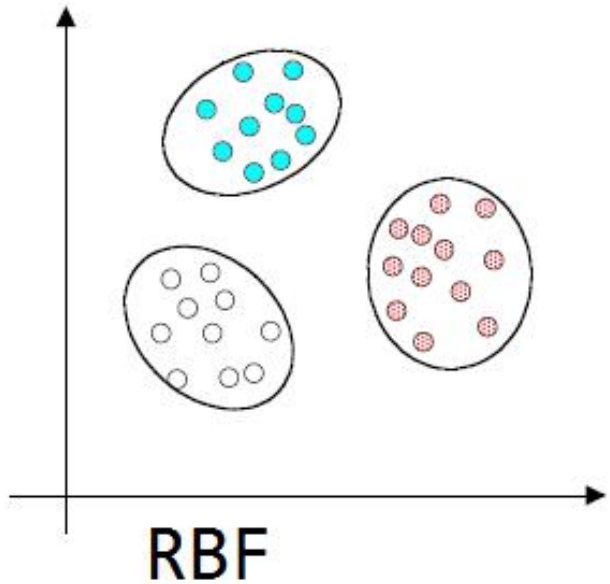
- Incrementally add RBFs
  1. Start by one hidden neuron; its center is the data point with lowest error
  2. Update the weights and calculate the errors
  3. If the error is less than the desired error then
     - Incrementally add basis functions, one-by one
     - Each time choose a center form the remaining data points
     - Go to step 2
  4. Otherwise stop the training


Start with an RBFN with one hidden neuron; Select its center from the data point with lowest cost
Update the weights by applying all pattern data & calculate E
Add new hidden neuron; Select its center from the remaining data points with lowest cost
No
E<E_goal
YES
END

RBF Applications

- **Classification**
  - Suppose we have a data set that falls into three classes:



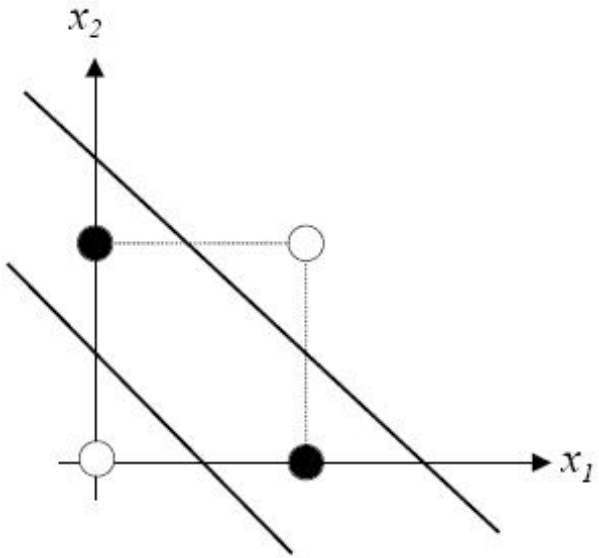


  - An MLP would separate the classes with hyper-planes in the input planes
  - RBF model the separate class distributions by localized basis functions

Example: XOR Problem

- Single layer perceptron with step or sigmoidal activation functions can not form the correct outputs, since they can only generate a single decision boundary.
- $\therefore$ We had introduced MLP (an extra layer)

p	x_1	x_2	t
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0



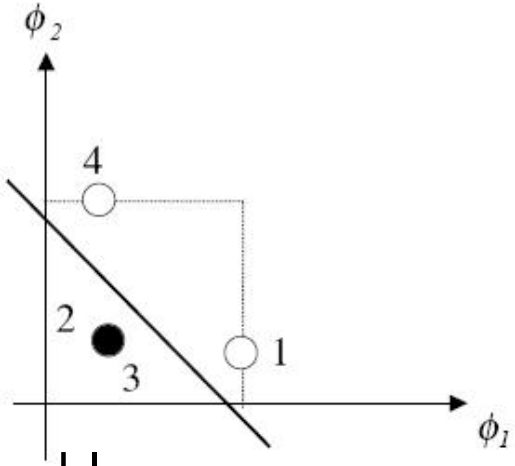
- Using RBFN: The Gaussian activation function is

$$\phi_j = exp(-\frac{M}{d_{max}^2}\|x - c_j^2\|)$$

- There are 4 patterns and 2 classes $\rightsquigarrow$ Choose $M = 2$
- Choose $\mu_1 = (0, 0)$ and $\mu 2 = (1, 1)$ as center of basis functions $\rightsquigarrow$ $d_{max} = \sqrt{2}$
- So the two basis functions are

$$\phi_1 = exp(-\|x - \mu_1\|), \quad \mu_1 = (0, 0)$$
$$\phi_2 = exp(-\|x - \mu_2\|), \quad \mu_2 = (1, 1)$$

p	x_1	x_2	ϕ_1	ϕ_2
1	0	0	1.0000	0.1353
2	0	1	0.3678	0.3678
3	1	0	0.3678	0.3678
4	1	1	0.1353	1.0000



- the patterns are now linearly separable
- Now the output weights should be adjusted:

$$y(x) = w_1\phi_1 + \phi_2 w_2 - \theta$$

- Considering the patterns:

$$\begin{aligned} 0 &= 1w_1 + 0.1353w_2 - 1\theta \\ 1 &= 0.3678w_1 + 0.3678w_2 - 1\theta \\ 1 &= 0.3678w_1 + 0.3678w_2 - 1\theta \\ 0 &= 0.1353w_1 + 1w_2 - 1\theta \end{aligned}$$

- Therefore $w_1 = w_2 = -2.5018, \theta = -2.8404$

- Other applications of RBFN are in
  - Model prediction
  - speech-hand writing recognition
  - control for high dimensional and highly nonlinear systems
  - image processing
  - time series analysis
  - medical diagnosis
- **Comparison of RBFN and MLP**
  - Similarities
    1. They are both non-linear feed-forward networks.
    2. They are both universal approximators.
    3. They are using in similar application areas.
  - There always exists an RBFN to accurately mimic a specified MLP, or vice versa
  - Differences
    1. An RBFN (in its most basic form) has a single hidden layer. But an MLP may have one or more hidden layers (AN RBF proposed by Lapendes (1991) has two hidden layers where the second hidden layer and output layer are both linear.)

Differences

2. MLPs construct global approximations to nonlinear I/O mappings $\rightsquigarrow$ they are capable of generalization in region of input space that little or no training data are available. But RBF networks tend to use localized nonlinearities (Gaussian func.) at the hidden layer to construct local approximations $\rightsquigarrow$ they can learn fast.
3. Due to local approximation of RBF, they may require larger number of hidden nodes to span the input space adequately as compared to MLP
4. In MLP the computation nodes in different layers share a common neural model (not necessarily the same activation function). But in RBFN, the hidden nodes (basis funcs) operate very different and have different purpose to the output nodes.
   - In RBFN, the argument of each hidden unit activation func is the distance between the input and the weights (RBF centers), But in MLPs it is the inner product of input and the weights


S. Chen, C. F. N. Cowan, and P.M. Grant.
Orthogonal least squares learning algorithm for radial basis function networks.
IEEE Transactions on Neural Networks, vol. 2, no. 2 , pages 302–309, March 1991.